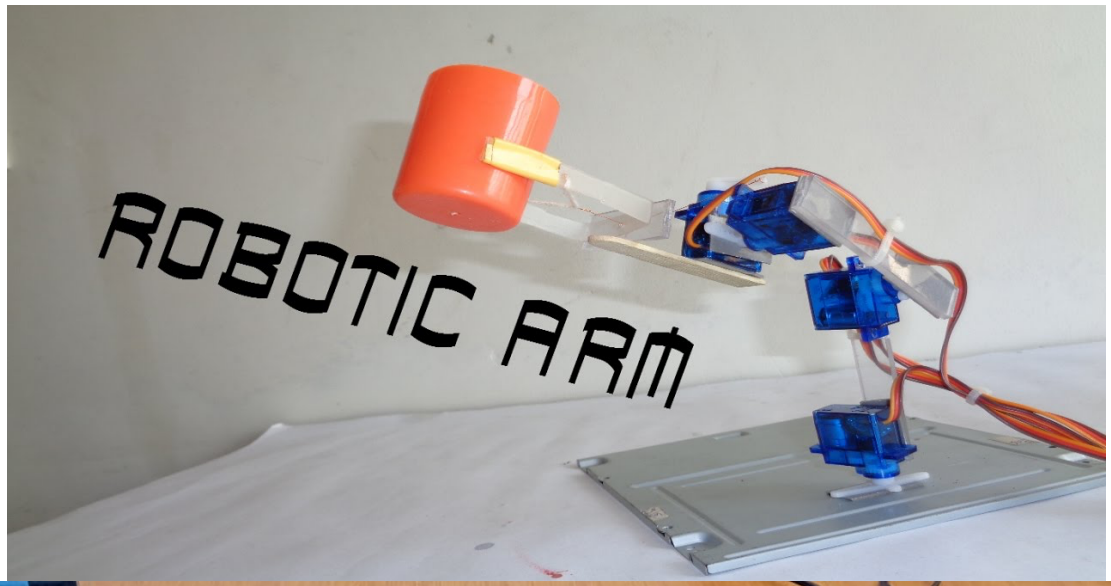
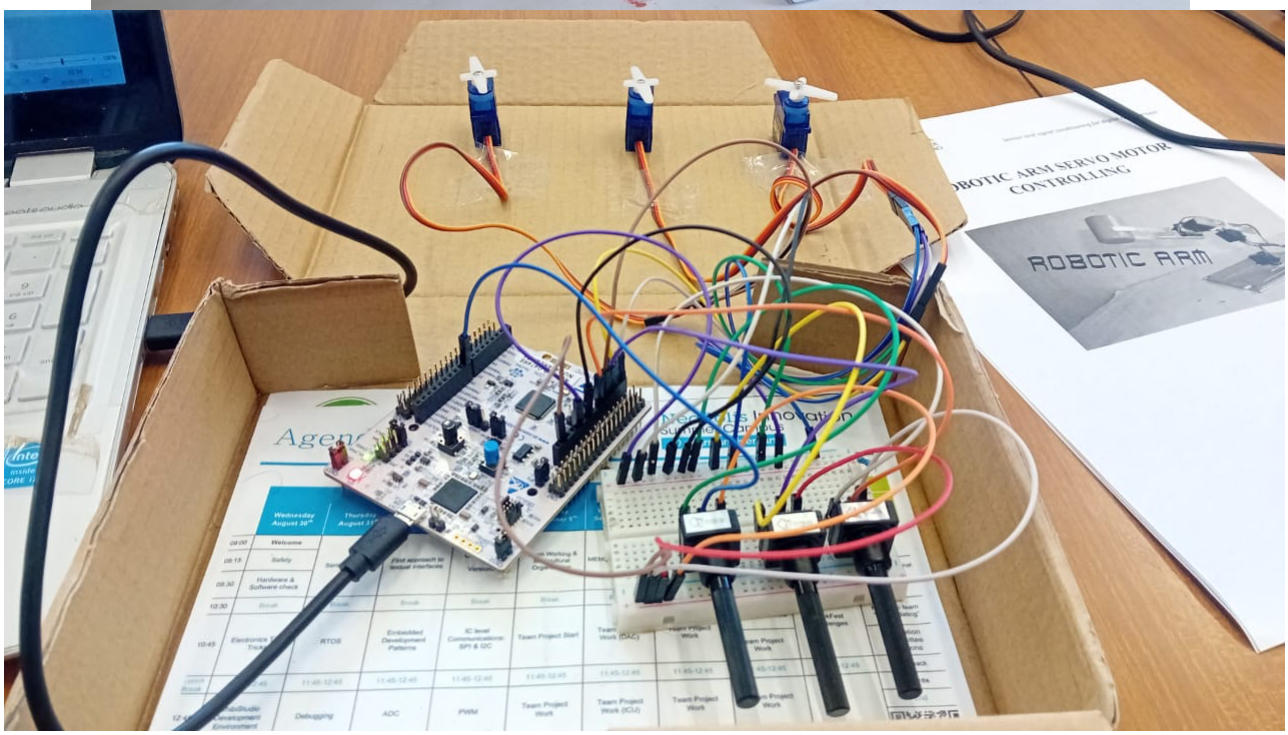
ROBOTIC ARM SERVO MOTOR CONTROLLING





Actuator used

Type of Actuator
SERVO MOTOR 9G 3 PIECES
Potentiometer 5kohm *3 PIECES*

Hardware used

Type of hardware
Ex. ST Nucleo G474RE

Software used

Type of software → *Ex. STM32 CubeMX*

Group composed by:

YASIN SHADROUH - 0780715
HAMED NAHVI - 0780707

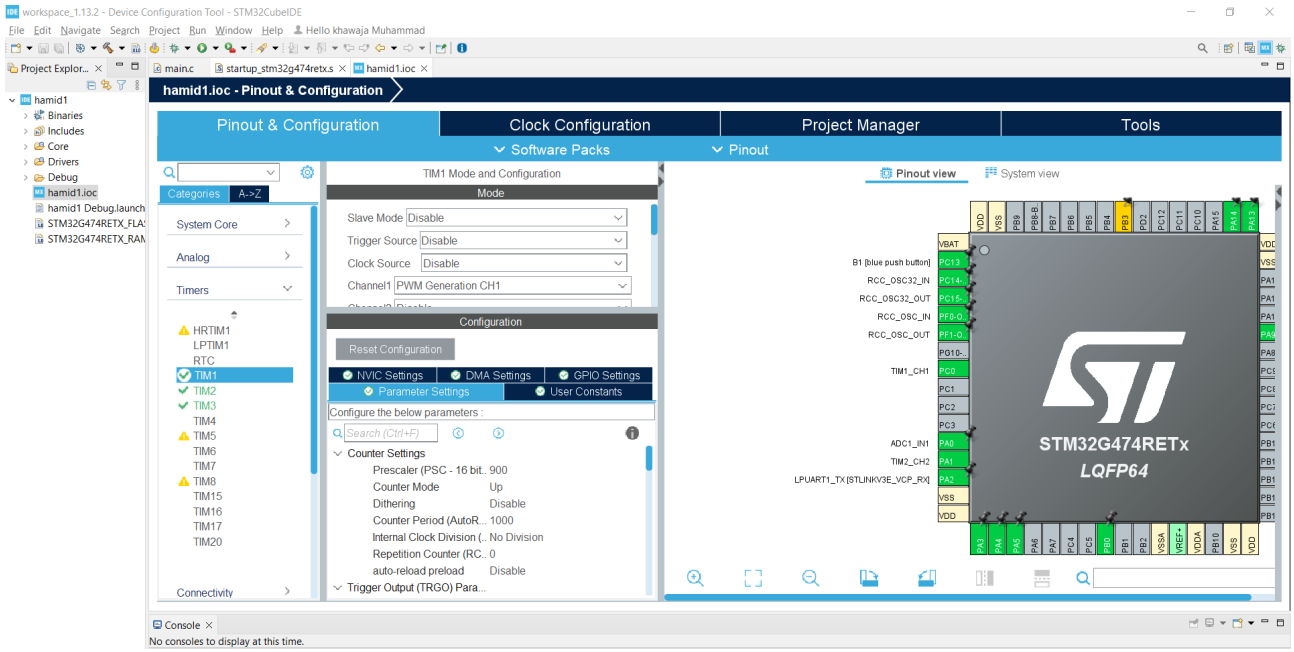
Description of the project idea:
Robotic arm that moves by chain of motors gives the freedom to perform action at any specific position. It basically works in industries having the distribution of objects from one place to another place for instance move object from table to conveyer belt.
This project is also cost effective and user friendly.

Actuator description (how it works):
We used the servo motor 9g for doing the operation of robotic arm that allow the precise control of angular motion. The output torque of the motor is 1.8kg/cm means it can lift 1 kg weight at a distance of 1cm.

Simply this project work by using the potentiometer when we spin the potentiometer knob the servo motor will rotate from 0-90 degrees

In order control the servo motor we will supply the pulse frequency of 50 Hz and we will change its width which will results in rotation of servo motor.
we will set the timer clock to 45MHz in clock configuration settings and we need frequency output 50Hz we will change its width result in rotation of servo motor.
Divide the 45Mhz by 50 we will get the value nine hundred thousand (900000)

we will distribute this value to prescalar value and ARR value
Prescalar = 900
ARR value =1000

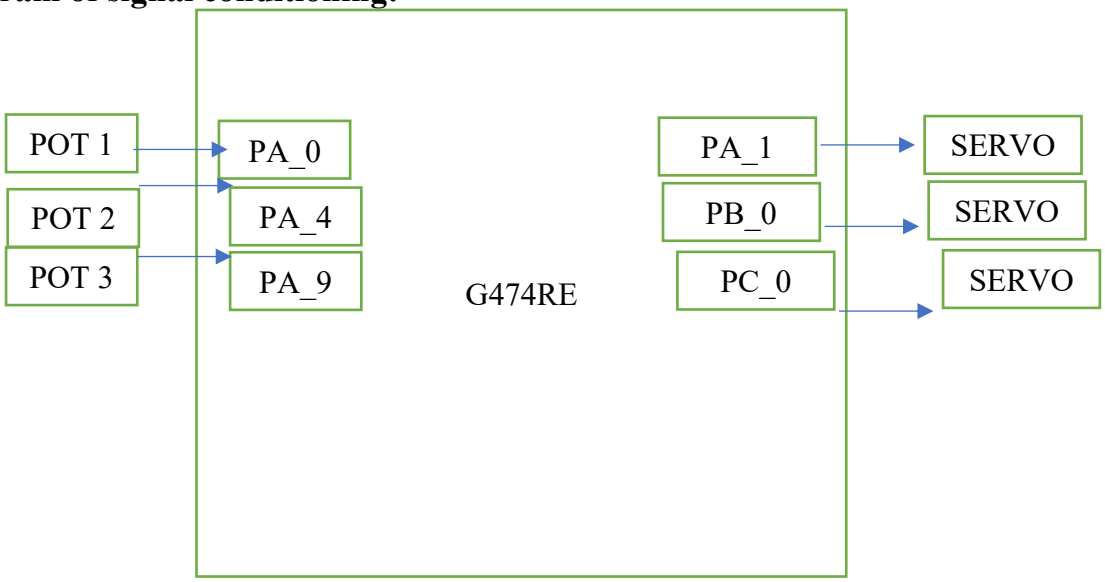


We know the formula t=1/f → t=1/50= 20ms
So 0.5ms is the 2.5% of 20ms
So if we move the value to CCR1 25 then servo move to the position 0 degrees, it means at 0.5 seconds the position of servo is 0 degrees
Likewise at 1.5ms the value of CCR1 will be 75 and the position of servo will be 90 degrees

and so on.

0 degree=cccr1 25= 0.5ms and 90 degree= ccr1 75= 1.5ms (means add 50hz to ccr1 will add 90 degree)

Wiring diagram of signal conditioning:



Description of used hardware:
G474RE ELECTRONIC CONTROLLER
POTENTIOMETER 5 KOHM USED FOR ADC CONVERSION
SERVO MOTOR 9G FOR POSITION SELECTION
Peripherals initialized in CubeMX:
A0 → PA0→ POTENTIOMETER CHANNEL ADC1_IN1
A1 → PA_1→ SERVO 1 TIM2_CH2
A2 → PA-4→ POTENTIOMETER CHANNEL ADC2_IN1

A3 → PB_0→ SERVO 2 TIM3_CH3

D8 → PA-09→ POTENTIOMETER CHANNEL ADC5_IN1

A5 → PC_0→ SERVO 3 TIM1_CH1

CLOCK CONFIGURATION HCLK = 45MHZ
Prescalar value of Servo1, Servo 2 and Servo3 is = 900
ARR value of Servo1, Servo 2 and Servo3 is = 1000

$$f_{\text{timer}} = \frac{f_{\text{HCLK}}}{(\text{Prescaler} + 1)}$$

Substituting **HCLK = 45 MHz** and **PSC = 900**:

$$f_{\text{timer}} = \frac{45,000,000}{(900+1)} = \frac{45,000,000}{901} \approx 49,945 \text{ Hz} \approx 50kHz$$

So, after applying the prescaler, the timer runs at **50 kHz**.

Code written in CubeIDE:

```
/* USER CODE BEGIN 0 */
uint16_t readValue;   // servo 1 potentiometer reading 0-4096

uint16_t Degree;      // servo 1 provide the degrees 0-180
uint16_t Angle;       // servo 1 putting the value to compare register CCR

uint16_t readValue2;   // servo 2  potentiometer reading 0-4096

uint16_t Degree2;      // servo 2   provide the degrees 0-180
uint16_t Angle2;       // servo 2   putting the value to compare register CCR

uint16_t readValue3;   // servo 3   potentiometer reading 0-4096

uint16_t Degree3;      // servo 3   provide the degrees 0-180
uint16_t Angle3;       // servo 3   putting the value to compare register CCR

/* USER CODE END 0 */

/* USER CODE BEGIN 2 */
  HAL_ADC_Start(&hadc1);                              //Servo 1
  HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_2);            //Servo 1

  HAL_ADC_Start(&hadc2);                              //Servo 2
    HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_3);            //Servo 2
```

```
    HAL_ADC_Start(&hadc5);                              //Servo 3
      HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_1);           //Servo 3

/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1, 1000);
readValue=HAL_ADC_GetValue(&hadc1);
Degree=readValue * (180)/( 4096);           0-180 scaling with 0-4096
Angle= 25 + (100)*(Degree)/(180);           25-100 CCR2(ANGLE) scaling with 0-180 degree

htim2.Instance->CCR2 = Angle;


HAL_ADC_Start(&hadc2);
HAL_ADC_PollForConversion(&hadc2, 1000);
readValue2=HAL_ADC_GetValue(&hadc2);
Degree2=readValue2 * (180)/(4096);          0-180 scaling with 0-4096
Angle2= 25 + (100)*(Degree2)/(180);        25-100 CCR3(ANGLE2)scaling with 0-180 degree

htim3.Instance->CCR3 = Angle2;




HAL_ADC_Start(&hadc5);
HAL_ADC_PollForConversion(&hadc5, 1000);
readValue3=HAL_ADC_GetValue(&hadc5);
Degree3=readValue3 * (180)/(4096);         0-180 scaling with 0-4096
Angle3= 25 + (100)*(Degree3)/(180);       25-100 CCR3(ANGLE2)scaling with 0-180 degree

htim1.Instance->CCR1 = Angle3;
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Detailed Explanation of the Code in CubeIDE for Servo Motor Control Using ADC & PWM

This code reads **analog values from potentiometers** using **ADC (Analog-to-Digital Conversion)** and converts them into **PWM signals** for controlling **three servo motors**. The servos are controlled using **timers (TIMx) and PWM signals** with duty cycles that correspond to **servo angles (0° to 180°).**

1. Variable Definitions (USER CODE BEGIN 0)

```
uint16_t readValue;   // Servo 1 potentiometer reading (0-4096)
uint16_t Degree;      // Servo 1 angle in degrees (0-180)
uint16_t Angle;       // Servo 1 CCR (PWM duty cycle value)

uint16_t readValue2;  // Servo 2 potentiometer reading (0-4096)
uint16_t Degree2;     // Servo 2 angle in degrees (0-180)
uint16_t Angle2;      // Servo 2 CCR (PWM duty cycle value)

uint16_t readValue3;  // Servo 3 potentiometer reading (0-4096)
uint16_t Degree3;     // Servo 3 angle in degrees (0-180)
uint16_t Angle3;      // Servo 3 CCR (PWM duty cycle value)
```

• **readValue**: Stores the ADC value from the potentiometer (0-4096).

• **Degree**: Maps the ADC value to a servo angle (0-180°).

• **Angle**: Maps the angle to the **CCR** (Capture/Compare Register) for PWM control.

Each servo motor has its **own set of variables**.

2. Initializing ADC and PWM (USER CODE BEGIN 2)

```
HAL_ADC_Start(&hadc1);                          // Start ADC for Servo 1
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);       // Start PWM for Servo 1

HAL_ADC_Start(&hadc2);                          // Start ADC for Servo 2
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);       // Start PWM for Servo 2

HAL_ADC_Start(&hadc5);                          // Start ADC for Servo 3
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);       // Start PWM for Servo 3
```

• **Starts ADC conversions** for all three servo motors.

• **Starts PWM timers** on specific channels:

• **Servo 1 → TIM2, Channel 2**

• **Servo 2 → TIM3, Channel 3**

• **Servo 3 → TIM1, Channel 1**

3. Infinite Loop for Servo Control (while (1))

Inside the loop, the system continuously reads potentiometer values, converts them to degrees, and updates the PWM signal to control servo motors.

3.1 Reading and Controlling Servo 1

```
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1, 1000);
readValue = HAL_ADC_GetValue(&hadc1);
Degree = readValue * (180) / (4096);        // Convert ADC value to 0-180 degrees
Angle = 25 + (100) * (Degree) / (180);      // Map 0-180 degrees to CCR values
(25-125)

htim2.Instance->CCR2 = Angle;               // Set PWM duty cycle for Servo 1
```

• **Start ADC conversion** for **Servo 1** (HAL_ADC_Start()).

• **Wait until ADC completes** (HAL_ADC_PollForConversion()).

• **Read ADC value** (0-4096) and **convert it to degrees (0-180°)**.

• **Map the degree value to the PWM duty cycle** (CCR2 = 25-125).

• **Update the PWM register** (CCR2) of **TIM2**.

3.2 Reading and Controlling Servo 2

```
HAL_ADC_Start(&hadc2);
HAL_ADC_PollForConversion(&hadc2, 1000);
readValue2 = HAL_ADC_GetValue(&hadc2);
Degree2 = readValue2 * (180) / (4096);    // Convert ADC value to degrees
Angle2 = 25 + (100) * (Degree2) / (180);  // Map degrees to CCR values (25-125)

htim3.Instance->CCR3 = Angle2;            // Set PWM duty cycle for Servo 2
```

• **Similar process for Servo 2** using ADC2 and **TIM3, Channel 3**.

3.3 Reading and Controlling Servo 3

```
HAL_ADC_Start(&hadc5);
HAL_ADC_PollForConversion(&hadc5, 1000);
readValue3 = HAL_ADC_GetValue(&hadc5);
Degree3 = readValue3 * (180) / (4096);   // Convert ADC value to degrees
Angle3 = 25 + (100) * (Degree3) / (180); // Map degrees to CCR values (25-125)

htim1.Instance->CCR1 = Angle3;           // Set PWM duty cycle for Servo 3
```

• **Same process for Servo 3**, using ADC5 and **TIM1, Channel 1**.

4. How the Mapping Works

The conversion between **ADC values → Degrees → PWM duty cycle** is crucial.

4.1 ADC to Degrees Mapping

```
Degree = readValue * (180) / (4096);
```

• Since ADC values range from **0-4096**, and we need **0-180°**, the formula scales the value linearly.

4.2 Degrees to PWM Duty Cycle Mapping

```
Angle = 25 + (100) * (Degree) / (180);
```

• **25 corresponds to 0.5ms pulse width (0°).**

• **125 corresponds to 2.5ms pulse width (180°).**

• **Intermediate values control angles between 0°-180°.**

4.3 PWM Pulse Width Calculation

• Since the **PWM period = 20ms (ARR = 1000)**, the pulse width must be:

• **0.5ms (2.5% of 20ms) → 0° → CCR = 25**

• **1.5ms (7.5% of 20ms) → 90° → CCR = 75**

• **2.5ms (12.5% of 20ms) → 180° → CCR = 125**

This ensures correct **servo positioning**.

5. Summary of the Code

Component	Description
ADC	Reads potentiometer values (0-4096)
PWM Timers	Generate PWM signals for servo control
CCR Registers	Control pulse width to set servo angle
Angle Mapping	Maps ADC values (0-4096) to servo angles (0-180°)

How It Works

1. **ADC reads** the potentiometer position.

2. The **value is mapped** to **0-180°**.

3. The **angle is converted to PWM CCR values (25-125)**.

4. The **PWM signal is updated** to rotate the servo.

5. The loop runs infinitely, continuously **updating servo positions**.

Final Thoughts

• This **efficiently controls three servos** using potentiometers.

• **PWM timing is properly configured** to meet servo specifications